CS 540: Introduction to Artificial Intelligence

Homework Assignment #3

Assigned: Tuesday, October 8
Due: Sunday, October 20

Hand-in Instructions

This homework assignment includes two written problems and a programming problem in Java. Hand in all parts electronically to your Canvas assignments page. For *each* written question, submit a single **pdf** file containing your solution. Handwritten submissions *must* be scanned. **No photos or other file types allowed**. For the programming question, submit a zip file containing *all* the Java code necessary to run your program, whether you modified provided code or not.

Submit the following **three** files (with exactly these names):

```
<wiscNetID>-HW3-P1.pdf
<wiscNetID>-HW3-P2.pdf
<wiscNetID>-HW3-P3.zip
```

For example, for someone with UW NetID crdyer@wisc.edu the first file name must be: `crdyer-HW3-P1.pdf`

Late Policy

All assignments are due **at 11:59 p.m.** on the due date. One (1) day late, defined as a 24-hour period from the deadline (weekday or weekend), will result in 10% of the total points for the assignment deducted. So, for example, if a 100-point assignment is due on a Wednesday and it is handed in between any time on Thursday, 10 points will be deducted. **For this homework only, you may turn it in at most one (1) day late, irrespective of whether or not you have free late days or not. In other words, no assignment can be turned in later than Monday, October 21 at 11:59 p.m.** Written questions and program submission have the same deadline. A total of three (3) free late days may be used throughout the semester without penalty. Assignment grading questions must be discussed with a TA within one week after the assignment is returned.

Collaboration Policy

You are to complete this assignment individually. However, you may discuss the general algorithms and ideas with classmates, TAs, peer mentors and instructor in order to help you answer the questions. But we require you to:

- not explicitly tell each other the answers
- not to copy answers or code fragments from anyone or anywhere
- not to allow your answers to be copied
- not to get any code from the Web

Problem 1. Unsupervised Learning by Clustering [20 points]

Consider the following information about *distances* (in kilometers) between pairs of 9 cities:

	TYO	DEL	SHA	BJ	MUM	OSA	DHK	CCU	SEL
TYO	0	5847	1767	2099	6740	403	4894	5138	1156
DEL	5847	0	4243	3777	1163	5476	1425	1304	4686
SHA	1767	4243	0	1070	5039	1364	3161	3403	871
BJ	2099	3777	1070	0	4756	1777	3026	3266	956
MUM	6740	1163	5039	4756	0	6355	1895	1664	5608
OSA	403	5476	1364	1777	6355	0	4500	4744	821
DHK	4894	1425	3161	3026	1895	4500	0	244	3793
CCU	5138	1304	3403	3266	1664	4744	244	0	4038
SEL	1156	4686	871	956	5608	821	3793	4038	0

The (latitude, longitude) *locations* of these cities are: TYO (35.7, 139.7), DEL (28.7, 77.2), SHA (31.2 121.5), BJ (39.9, 116.4), MUM (19.1, 72.9), OSA (34.7, 135.5), DHK (23.8, 90.4), CCU (22.6, 88.4), and SEL (37.6, 127).

(a) [10] Perform (manually) **hierarchical agglomerative clustering** using *single-linkage* and the distance data in the above table.

(i) [8] Show the resulting dendrogram.

(ii) [2] What clusters of cities are created if you want 3 clusters?

(b) [10] Show the results of *one* (1) iteration of ***k*-means clustering** assuming $k = 2$ and the initial cluster centers are $c_1 = (38.0, 127.0)$ and $c_2 = (27.0, 117.0)$.

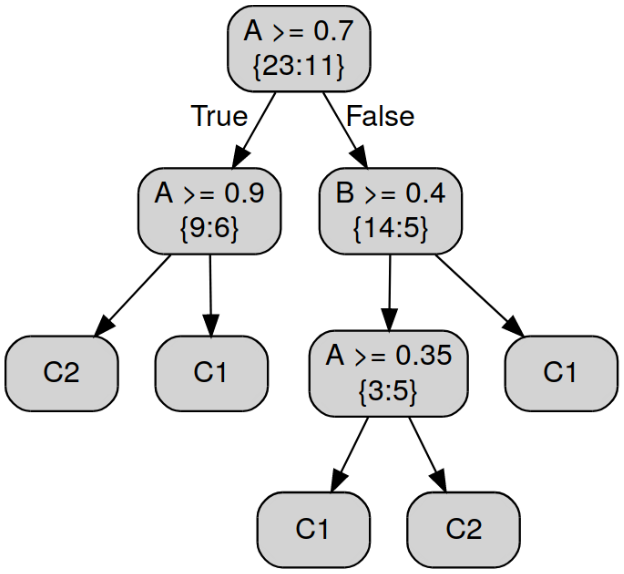
(i) [3] Give the list of cities in each of the initial 2 clusters.

(ii) [4] Give the coordinates of the new cluster centers.

(iii) [3] Give the list of cities in the 2 clusters based on the new cluster centers computed in (ii).

Problem 2. Decision Tree Pruning [15 points]

A given classification task has two classes, C1 and C2. There are two continuous, real-valued attributes called A and B. Given a set of training examples (aka instances), the following decision tree was built:



Each non-leaf node represents a test using one of the attributes. The numbers in braces at each non-leaf node correspond to the number of training examples that reach that node, where {*x*:*y*} means that a total of *x*+*y* training examples reached a node, *x* of them were in class C1 and *y* of them were in class C2. Each leaf node contains the class associated with all examples that reach that node.

Given the above tree, you are provided with the following Tuning Set:

A	B	Class
0.5	0.27	C1
0.41	0.54	C2
0.95	0.3	C2
0.25	0.62	C1
0.32	0.81	C2

Now answer the following questions:

(a) [3] What is the classification accuracy of the Tuning Set using the given decision tree?

(b) [10] Show the classification accuracy on the Tuning Set after removing each non-leaf node (including the root) and replacing it with a leaf node that has the majority class of the training examples associated with that node. Since there are four non-leaf nodes in the above decision tree, your answer should show **four** new decision trees as a result of removing each of the four non-leaf nodes independently.

(c) [2] Based on your answers in (b), which non-leaf node should be pruned first?

Problem 3: Binary Decision Trees [65 points]

In this problem you are to implement a program that builds a binary decision tree for numerical attributes, and binary classification tasks. By binary tree, we mean that every non-leaf node of your tree will have exactly two children. Each node will have a selected attribute and an associated threshold value. Instances (aka examples) that have an attribute value less than or equal to the threshold belong to the left subtree of a node, and instances with an attribute value greater than the threshold belong to the right subtree of a node. The programming part requires building a tree from a training set and classifying instances of both the training set and the testing set with the learned decision tree. Code has been provided for printing your decision tree in a specific format. You may change the code if you wish, but the output must be in **exactly** the same format that the provided function produces.

We have provided some skeleton code to help you on this problem. While you are not required to use the code, there are some aspects of your submission that you must conform to. The requirements of your submitted code are:

1. You must submit at least one java file named `HW3.java`, and it must contain your homework's static main method.

2. Your `HW3.java` file should accept 4 arguments, specified by the command:

```
java HW3 <train file> <test file> <maximum instances per leaf> <maximum depth>
```

3. Your `HW3.java` file should be able to read data in the format we specify and print the correct output for each of the modes. Your output must be exactly (including matching whitespace characters) the same as the example output that we have provided to you. Mismatched or mis-formatted output will be marked as incorrect.

4. Any supporting java files that you use with your code must be submitted with your `HW3.java` file.

In addition to constraints on the program files, there are some simplifying assumptions that you can use when developing your code:

1. All attributes will be (numerical) integer valued and have values from 1 to 10. There will be no categorical attributes or non-integer real-valued attributes in the set of training or test instances.

2. Splits on integer-valued attributes are binary (i.e., each split creates a <= set and a > set).

3. An attribute can be split on multiple times in the decision tree.

4. All attributes will appear in the same order for every instance (a row in the data file) and will always be separated by commas only.

5. The first column of every row in the input data file contains the ID and will be discarded. The last column contains the class label (2 or 4) for that specific instance. The skeleton code provided removes the first column and converts the class labels to 0 or 1 (2 to 0 and 4 to 1) when reading the file.

Aspects of the Tree

Your decision tree must have certain properties so that the program output based on your decision tree implementation matches our own. The constraints are:

1. The attribute values of your instances are integers, but the information gain calculation must be done using doubles. Remember to use the convention that **$0 \log_2 0 = 0$** when computing the information gain.

2. At any non-leaf node in the tree, the left child of a node represents instances that have an attribute value **less than or equal to** (<=) the threshold specified at the node. The right child of a node represents instances that have an attribute value **greater than** (>) the threshold specified at the node.

3. When selecting the attribute at a decision tree node, first find the best (i.e., highest information gain) threshold for each attribute by evaluating multiple candidate split thresholds. The candidate splits are integers in {1, 2, …, 9} for each attribute for this homework. Once the best candidate split is found for each attribute, choose the attribute that has the highest information gain (among the ones strictly larger than 0). If there are multiple attributes with the same information gain, split on the attribute that appears **earliest** in the list of attribute labels. If an attribute has multiple split thresholds producing the same best information gain, select the threshold with **lowest** integer value.

4. When creating a decision tree node, if the number of instances belonging to that node is less than or equal to the "maximum instances per leaf", or if the depth is equal to the "maximum depth" (the root node has depth 0), or if the maximum information gain is 0, then a leaf node must be created. The label assigned to the node is the majority label of the instances belonging to that node. If there are an equal number of instances labeled 0 and 1, then the **label 1** is assigned to the leaf.

Description of Program Outputs

We will test your program using several training and testing datasets using the command line format:

```
java HW3 <train file> <test file> <maximum instances per leaf> <maximum depth>
```

where `<train file>` and `<test file>` are the names of the training and testing datasets, formatted to the specifications described later in this document, and `<maximum instances per leaf>` and `<maximum depth>` are **strictly positive** integers. The output prints a decision tree built from the training set followed by the classification for each example in the testing set (either 0 or 1), and the accuracy (rounded to 2 decimal in percentage, a simple String.format("%.2f") should be fine to use) on the testing set. The format of the printed decision tree is provided in our skeleton code, we highly recommend that at least copy paste our print method in the skeleton code.

Data

The dataset we've provided comes from the Wisconsin Breast Cancer dataset and can be found at the UCI machine learning repository:
https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Original%29

The first column is an ID, not an attribute, and therefore is *not* used to construct the decision tree. The next nine columns are attributes, named X0, X1, …, X8, that have integer values from 1 to 10. The last column contains the class label (Y = 2 for benign, Y = 4 for malignant). Each row corresponds to one sample from a patient. Rows with missing data (rows that contain "?") are removed by the provided skeleton code. The training and test sets used to grade your code will *only* contain selected rows from this data set.

This means, for the purpose of this homework, you may use the special property that attributes X1 to X9 are integers 1 to 10, and the label Y is converted to 0 or 1 correctly when reading the input file.

We have provided input and output for the following three test cases:

```
java HW3 train_1.data test_1.data 10 10 -> output_1.txt
java HW3 train_2.data test_2.data 10 1 -> output_2.txt
java HW3 train_3.data test_3.data 1 10 -> output_3.txt
```

You can download the complete data set from the UCI machine learning repository for additional training and test data if you desire.

Deliverables

Put all `.java` files needed to run your program, including ones you wrote, modified or were given and are unchanged, into a folder called `<wiscNetID>-HW3-P3` **Do not include training and test data** in your submission. Compress this folder to create `<wiscNetID>-HW3-P3.zip` and upload it to Canvas. For example, for someone with UW NetID `crdyer@wisc.edu` the file name must be: `crdyer-HW3-P3.zip`

Note on Splitting on Real-Valued Attributes (Not Required for this Homework)

For this homework, since the attributes are integers in a small range between 1 and 10, the candidate thresholds {1, 2, 3, …, 9} is a small set. In this case, it is simplest to just compute the information gain for all possible thresholds and find the one corresponding to the largest information gain (and use the smallest threshold in case of ties). In practice (i.e., NOT this homework), with real-valued attributes or with a larger range, to find candidate thresholds to use for an attribute, one good way is to sort the set of instances at the current node by attribute value, and then find consecutive instances that have different class labels. A difference in information gain can only occur at these boundaries. For each pair of consecutive instances that have different classes, you can then compute the average value of these two consecutive instances as a candidate threshold value.